

## DCODETX.S

```
; 4/11/95
;with stop
;!!!!!!! note:for z8604 with external EEPROM & RS232 !!!!!!!!
```

## EQUATE STATEMENTS

```
XRGRPF      .equ    0f0H      ; expanded reg group F (WDT,SMR,PCON)
XRGRP0      .equ    00H       ; expanded reg group 0 (ports)
S1B39       .equ    00000000b  ; B39 value for S1
S2B39       .equ    00000001b  ; B39 value for S2
S3B39       .equ    00000010b  ; B39 value for S3
S1          .equ    00000100b  ; P32 S1 mask for Z86C04
S2          .equ    00001000b  ; P33 S2 mask for Z86C04
S3          .equ    00000010b  ; P31 S3 mask for Z86C04
smr         .equ    0bH       ; stop mode recovery
csh         .equ    00000100b  ;P22 chip sel hi for 93c46
cs1         .equ    11111011b  ;P22 chip sel lo for 93c46
clockh      .equ    00000010b  ;P21 clk hi for 93c46
clockl      .equ    11111101b  ;P21 clk lo for 93c46
doh         .equ    00000001b  ;P20 data out hi for 93c46
dol         .equ    11111110b  ;P20 data out lo for 93c46
csport      .equ    P2        ;chip sel port 93c46
dioport     .equ    P2        ;data i/o port 93c46
clkport     .equ    P2        ;clk port 93c46
```

## CONTROL REG AND INITIAL VALUES

```
STACKTOP    .equ    07FH      ; start of the stack
STACKEND    .equ    070H      ; end of the stack
GPR_INIT    .EQU    00H       ; init general purpose reg to 00H
RP_INIT     .EQU    00H       ; init register pointer to 00
IMR_INIT    .EQU    00000000B ; init intr mask reg (di)
IPR_INIT    .EQU    00001111B ; init intr priority reg
P01M_INIT   .EQU    00000100B ; init port 0&1 mode reg
P2M_INIT    .EQU    10010000B ; init port2 mode
P3M_INIT    .EQU    00000001B ; init port3 mode
PRE1_INIT   .EQU    00001011B ; init prescalar 1 reg
T1_INIT     .EQU    250D      ; init counter/timer 1 reg /200
TMR_INIT    .EQU    00000000B ; init timer mode reg
TMR_START   .EQU    00001100B ; start timer
P0_INIT     .EQU    00000000B ; init port0
P2_INIT     .EQU    00000000B ; init port2
P3_INIT     .EQU    00000000B ; init port3

SMR_INIT    .EQU    11111010B ; init SMR reg bit1 hi OTP Lo Emulato
r

PCON_INIT   .EQU    11111110B ; init Port control reg
```

## PREDEFINED CONTROL REG

```
;SPL      .equ    255      ; stack pointer
GPR       .equ    254      ; general purpose
;RP       .equ    253      ; register pointer
;FLAGS    .equ    252      ; cpu flags
```

A-1

## DCODETX.S

```

RS232ODELAY      .EQU    REGGRP10+10
RS232IDELAY      .EQU    REGGRP10+11
RS232CCOUNT      .EQU    REGGRP10+12
RS232PAGE        .EQU    REGGRP10+13
RSCCOUNT        .EQU    REGGRP10+14
RSSTART          .EQU    REGGRP10+15

RS232OS          .EQU    00000100B      ;RS232 output bit set
RS232OC          .EQU    11111011B      ;RS232 output bit clear
RS232OP          .EQU    P0              ;RS232 output port
RS232IP          .EQU    P2              ;RS232 input port
RS232IM          .EQU    00010000B      ;RS232 input mask
;*****
; GENERAL PURPOSE REGISTER GROUP 20H-2FH
;*****
REGGRP20         .equ     20H
TRC0             .equ     REGGRP20      ;Trinary Roll Code REG's LSB
TRC1             .equ     REGGRP20+1    ;Trinary Roll Code REG's
TRC2             .equ     REGGRP20+2    ;Trinary Roll Code REG's
TRC3             .equ     REGGRP20+3    ;Trinary Roll Code REG's
TRC4             .equ     REGGRP20+4    ;Trinary Roll Code REG's
TRC5             .equ     REGGRP20+5    ;Trinary Roll Code REG's
TRC6             .equ     REGGRP20+6    ;Trinary Roll Code REG's
TRC7             .equ     REGGRP20+7    ;Trinary Roll Code REG's
TRC8             .equ     REGGRP20+8    ;Trinary Roll Code REG's
TRC9             .equ     REGGRP20+9    ;Trinary Roll Code REG's
SYNC1           .equ     REGGRP20+10    ;sync pulse frame1
TRC10            .equ     REGGRP20+11    ;Trinary Roll Code REG's
TRC11            .equ     REGGRP20+12    ;Trinary Roll Code REG's
TRC12            .equ     REGGRP20+13    ;Trinary Roll Code REG's
TRC13            .equ     REGGRP20+14    ;Trinary Roll Code REG's
TRC14            .equ     REGGRP20+15    ;Trinary Roll Code REG's

trc0             .equ     r0             ;Trinary Roll Code REG's LSB
trc1             .equ     r1             ;Trinary Roll Code REG's
trc2             .equ     r2             ;Trinary Roll Code REG's
trc3             .equ     r3             ;Trinary Roll Code REG's
trc4             .equ     r4             ;Trinary Roll Code REG's
trc5             .equ     r5             ;Trinary Roll Code REG's
trc6             .equ     r6             ;Trinary Roll Code REG's
trc7             .equ     r7             ;Trinary Roll Code REG's
trc8             .equ     r8             ;Trinary Roll Code REG's
trc9             .equ     r9             ;Trinary Roll Code REG's
sync1           .equ     r10            ;sync pulse frame1
trc10            .equ     r11            ;Trinary Roll Code REG's
trc11            .equ     r12            ;Trinary Roll Code REG's
trc12            .equ     r13            ;Trinary Roll Code REG's
trc13            .equ     r14            ;Trinary Roll Code REG's
trc14            .equ     r15            ;Trinary Roll Code REG's

;*****
; GENERAL PURPOSE REGISTER GROUP 30H-39H (3Ah-3FH reserved for stack)
;*****
REGGRP30         .equ     30H
TRC15            .equ     REGGRP30      ; Trinary Roll Code REG's
TRC16            .equ     REGGRP30+1    ; Trinary Roll Code REG's
TRC17            .equ     REGGRP30+2    ; Trinary Roll Code REG's
TRC18            .equ     REGGRP30+3    ; Trinary Roll Code REG's MSB
TRC19            .equ     REGGRP30+4    ; sync pulse frame0
SYNC0            .equ     REGGRP30+5    ; sync pulse frame0

```

## DCODETX.S

```

RCMIR0      .equ    REGGRP30+6      ; RC mirrored less LSB
RCMIR1      .equ    REGGRP30+7      ; RC mirrored less
RCMIR2      .equ    REGGRP30+8      ; RC mirrored less
RCMIR3      .equ    REGGRP30+9      ; RC mirrored less MSB

trcl5       .equ    r0              ; Trinary Roll Code REG's
trcl6       .equ    r1              ; Trinary Roll Code REG's
trcl7       .equ    r2              ; Trinary Roll Code REG's
trcl8       .equ    r3              ; Trinary Roll Code REG's MSB
trcl9       .equ    r4              ; sync pulse frame0
sync0       .equ    r5              ; spare
rcmir0      .equ    r6              ; RC mirrored less LSB
rcmir1      .equ    r7              ; RC mirrored less
rcmir2      .equ    r8              ; RC mirrored less
rcmir3      .equ    r9              ; RC mirrored less MSB
;*****
; GENERAL PURPOSE REGISTER GROUP 40H-4FH
;*****
REGGRP40    .equ    40H              ;
XMTREG      .equ    REGGRP40         ;
LPCTR       .equ    REGGRP40+1       ;
XR00        .equ    REGGRP40+2       ;
XMTREG1     .equ    REGGRP40+3       ;
ACODEPTR    .equ    REGGRP40+4       ;
MTFLAG      .equ    REGGRP40+5       ;
DIVBY10     .equ    REGGRP40+6       ;
TRCPtr      .equ    REGGRP40+7       ;
TEMPH       .equ    REGGRP40+8       ;ee
TEMPL       .equ    REGGRP40+9       ;ee
TEMP        .equ    REGGRP40+10      ;ee
MTEMPH      .equ    REGGRP40+11      ;memory tem eeprom
MTEMPL      .equ    REGGRP40+12      ;memory tem eeprom
MTEMP       .equ    REGGRP40+13      ;memory tem eerom
SERIAL      .equ    REGGRP40+14      ;serial data to/from eeprom
ADDRESS     .equ    REGGRP40+15      ;eeprom address

xmtreg      .equ    r0              ;
lpctr       .equ    r1              ;
xr00        .equ    r2              ;
xmtreg1     .equ    r3              ;
acodeptr    .equ    r4              ;
mtflag      .equ    r5              ;
divby10     .equ    r6              ;
trcptr      .equ    r7              ;
temph       .equ    r8              ;
templ       .equ    r9              ;
temp        .equ    r10             ;
mtemph      .equ    r11             ;
mtempl      .equ    r12             ;
mtemp       .equ    r13             ;
serial      .equ    r14             ;
address     .equ    r15             ;
;
;*****
; GENERAL PURPOSE REGISTER GROUP 50H-5FH
;*****
REGGRP50    .equ    50H              ;
ACODE0BM    .equ    REGGRP50         ;
ACODE1BM    .equ    REGGRP50+1      ;

```

A-5

## DCODETX.S

```

ACODE2BM      .equ    REGGRP50+2      ;
ACODE3BM      .equ    REGGRP50+3      ;
ACODE4BM      .equ    REGGRP50+4      ;
ACODE5BM      .equ    REGGRP50+5      ;
ACODE6BM      .equ    REGGRP50+6      ;
ACODE7BM      .equ    REGGRP50+7      ;
ACODE8BM      .equ    REGGRP50+8      ;
ACODE9BM      .equ    REGGRP50+9      ;
ACODE10BM     .equ    REGGRP50+10     ;
ACODE11BM     .equ    REGGRP50+11     ;
ACODE12BM     .equ    REGGRP50+12     ;
ACODE13BM     .equ    REGGRP50+13     ;
ACODE14BM     .equ    REGGRP50+14     ;
ACODE15BM     .equ    REGGRP50+15     ;

acode0bm      .equ    r0              ;
acode1bm      .equ    r1              ;
acode2bm      .equ    r2              ;
acode3bm      .equ    r3              ;
acode4bm      .equ    r4              ;
acode5bm      .equ    r5              ;
acode6bm      .equ    r6              ;
acode7bm      .equ    r7              ;
acode8bm      .equ    r8              ;
acode9bm      .equ    r9              ;
acode10bm     .equ    r10             ;
acode11bm     .equ    r11             ;
acode12bm     .equ    r12             ;
acode13bm     .equ    r13             ;
acode14bm     .equ    r14             ;
acode15bm     .equ    r15             ;

;
; *****
; GENERAL PURPOSE REGISTER GROUP 60H-6FH
; *****
REGGRP60      .equ    60H              ;
ACODE16BM     .equ    REGGRP60         ;
ACODE17BM     .equ    REGGRP60+1       ;
ACODE18BM     .equ    REGGRP60+2       ;
ACODE19BM     .equ    REGGRP60+3       ;
RSFLAG        .equ    REGGRP60+4       ;
XMTFLAG       .equ    REGGRP60+5       ;
AC19          .equ    REGGRP60+6       ;
RCP           .equ    REGGRP60+7       ;
LPCNTRA       .equ    REGGRP60+8       ;
FRMCTRH       .equ    REGGRP60+9       ;
FRMCTRL       .equ    REGGRP60+10      ;
ATMP          .equ    REGGRP60+11      ;acode tmp storage
;acode_h      .equ    REGGRP60+12      ;acode rom pointerh
;acode_l      .equ    REGGRP60+13      ;acode rom pointerl
LPCTR1        .equ    REGGRP60+14      ;counter
APTR          .equ    REGGRP60+15      ;acode ram pointer

acode16bm     .equ    r0              ;
acode17bm     .equ    r1              ;
acode18bm     .equ    r2              ;
acode19bm     .equ    r3              ;
rsflag        .equ    r4              ;
xmtflag       .equ    r5              ;
ac19          .equ    r6              ;

```

A-6

## DCODETX.S

```

rcp                .equ    r7      ;
lpcntra            .equ    r8      ;
frmctrh            .equ    r9      ;
frmctrl            .equ    r10     ;
atmp               .equ    r11     ;acode tmp storage
acode              .equ    rr12    ;acode register pair
acode_h            .equ    r12     ;acode rom pointer h
acode_l            .equ    r13     ;acode rom pointer l
lpctl              .equ    r14     ;counter
aptr               .equ    r15     ;acode ram pointer
;
;*****
; MACROS
;*****
;
WDT                .macro
                  .byte    5fh
                  .endm
WDH                .macro
                  .byte    4fh
                  .endm
FILL               .macro
                  .byte    0FFh
                  .endm
;
;*****
;*****
;***** Interrupt Vector Table *****
;*****
;*****
;
                .org    0000H

                .word    000CH        ;IRQ0 P3.2
                .word    000CH        ;IRQ1, P3.3
                .word    000CH        ;IRQ2, P3.1
                .word    000CH        ;IRQ3, S/W generated
                .word    000CH        ;IRQ4, S/W generated
                .word    T1_INT       ;IRQ5,Timer T1
;
;*****
;*****
;***** START (poweron reset or stop mode) *****
;*****
;
                .page
                .org    000CH

start:
START:            di                    ; disable interrupts for init
                  WDT                   ; hit WDT
;
;*****
;* Internal RAM Test and Reset All RAM = ?? mS *
;*****
INIT:             srp    #XRGRPF        ;no,point to control group use stack

```

A-7

## DCODETX.S

```

        ld        r15,#4            ;r15= pointer (bottom of RAM)
write_again:  clr        @r15        ;write RAM(r5)=0 to memory
              inc        r15
              cp         r15,#7FH    ;top of ram 7F
              jr         ult,write_again
;
;*****
;      initialize registers
;*****
;
              srp        #REGGRP00    ; set the group
              ld         SMR,#SMR_INIT ; set smr reg
;
;*****
;      STACK INITIALIZATION
;*****
;
SETSTACK:    ld         spl,#STACKTOP ; set the start of the stack
;
;*****
;      TIMER INITIALIZATION
;*****
;
              ld         prel,#PRE1_INIT ; set the prescaler
              ld         t1,#T1_INIT      ; set the counter
              ld         tmr,#TMR_START   ; turn on the timer
;
;*****
;      PORT INITIALIZATION
;*****
;
              clr        P0             ; set port0 lo
              clr        P2             ; set port2 lo
              clr        P3             ; set port3 lo
              ld         p3m,#P3M_INIT   ; set port 3 mode
              ld         p2m,#P2M_INIT   ; set port 2 mode
              ld         p01m,#P01M_INIT ; set port 1 mode
;
;*****
;      INTERRUPT INITIALIZATION
;*****
;
SETINTERRUPTS: ld        ipr,#IPR_INIT ; set the priority for timer
;
;*****
;      initialize EEPROM by reading it
;*****
;
              CALL        READMEMORY      ;settle EE lines
;
;*****
;      MAIN LOOP
;*****
;
CKBUTTON1:   CALL        CKB1
              LD          ACODE19BM,AC19
              LD          RCPTR,RCP
;

```

A-8

## DCODETX.S

```

;*****
;      Get Rolling Code From EEPROM
;*****
;*****
;      EE_ADDRESS 11->RC10B,RC11B,RC12B,RC13B
;      EE_ADDRESS 13->RC20B,RC21B,RC22B,RC23B
;      EE_ADDRESS 15->RC30B,RC31B,RC32B,RC33B
;*****
INITPTRS:      srp      #REGGRP00
               add      RCPTR,#3      ;TOP OF RC RAM
               CP       RCPTR,#RC13B
               JR       nz,CKRC23
               LD       ADDRESS,#11    ;EE PTR
               JR       GETRCODE
CKRC23:        CP       RCPTR,#RC23B
               JR       nz,APTR15
               LD       ADDRESS,#13
               JR       GETRCODE
APTR15:        LD       ADDRESS,#15
GETRCODE:      LD       lpcntr,#2
GETRCODE1:     CALL     READMEMORY
               LD       @RCPTR,MTEMPH ;HI BYTE
               DEC      RCPTR
               LD       @RCPTR,MTEMPL ;LO BYTE
               DEC      RCPTR
               DEC      ADDRESS
               DJNZ     lpcntr,GETRCODE1 ;done?
               INC      RCPTR
;*****
;      Increment Rolling Code by 3
;*****
INCRBY3:       srp      #REGGRP10
               ADD      @rcptr,#3d      ;Add 3 to Rolling Code
               LD       bitptr,#3d
INCRNEXT:      INC      rcptr
               ADC      @rcptr,#0
               DJNZ     bitptr,INCRNEXT
;*****
;      Store updated Rolling Code in EEPROM
;*****
               CALL     CKB1      ;SAME BUTTON STILL
               CP       ACODE19BM,AC19 ;PRESSED?
               JP       nz,SCHTOPP

               srp      #REGGRP60
               ADD      ADDRESS,#2      ;START EEPROM ADDRESS
SAVRCODE:      LD       lpcntra,#2
SAVRCODE1:     LD       MTEMPH,@RCPTR ;hi byte
               DEC      RCPTR
               LD       MTEMPL,@RCPTR ;lo byte
               CALL     WRITEMEMORY
               DEC      RCPTR
               DEC      ADDRESS
               DJNZ     lpcntra,SAVRCODE1
               INC      RCPTR

```

A-9

## DCODETX.S

```

;*****
;      get ACODE0BM-ACODE18BM from eeprom
;*****
;
GETACODE:      srp      #REGGRP40
               ld        address,#9          ;highest eeprom addr
               ld        acodeptr,#ACODE18BM ;highest acode ram addr
               CALL      READMEMORY
               ld        @acodeptr,mtemp      ;hi byte
               DEC       acodeptr
               CP        acodeptr,#4Fh        ;4fh? done?
               JR        z,ACODONE
               ld        @acodeptr,mtemp1
               DEC       address
               djnz      acodeptr,GETACODE
ACODONE:
;*****
;      Mirror RCX0,1,2,3 into RCMIR0,1,2,3 and zero MSB
;*****
;
MIRROR:      srp      #REGGRP10
               ld        codeptr,#RCMIR3      ;RCMIR3 FIRST
NBYTE:      ld        bitptr,#08d            ; set bit counter to 7
SHIFT:      RL         @rcptr                ; shift RC into carry
               RRC       @codeptr             ; shift carry into mirror
               DJNZ      bitptr,SHIFT
               CP        codeptr,#RCMIR3      ; if RCMIR3 then
               JR        nz,NOTRC3
               AND       RCMIR3,#01111111b    ; set bit 7 RCMIR3 to 0
NOTRC3:      DEC       codeptr                ;next rcmir
               INC       rcptr
               CP        codeptr,#35H
               JR        nz,NBYTE
               sub       rcptr,#4
;*****
;      Trinary conversion & store in TRC0-TRC19
;*****
;
               srp      #REGGRP00            ;set reg pntr
;
               LD        lpcntr,#36H          ;ZERO OUT TRC PREVIOUS TRINARY #'s
ZAGN:      DEC       lpcntr
               CLR       @lpcntr
               CP        lpcntr,#20H
               JR        nz,ZAGN
;
               LD        TRCXX,#TRC19
               LD        RCPTR,#20
CALCTRNY:  CP        RCPTR,#01                ;calc trinary number
               JR        z,X3XX1
               CALL      ENTR3
               CP        RCPTR,#02            ;=2?
               JR        z,TRICONVXX
               SUB       RCPTR,#2
               LD        tcntr,RCPTR
               ADD       RCPTR,#2

```

A-10



## DCODETX.S

```

ADDAGN:      CALL    ENTR3A
              CALL    AD3XX          ;add to itself
              CALL    AD3XX
              CALL    XFER
              DJNZ    tcntr,ADDAGN    ;TCNTR=0?
              JR      TRICONVXX
X3XX1:       LD      x3xabcd,#01h
              clr     x3xabcd1
              clr     x3xabcd2
              clr     x3xabcd3

TRICONVXX:   SBC     RCMIR0,x3xabcd
              SBC     RCMIR1,x3xabcd1
              SBC     RCMIR2,x3xabcd2
              SBC     RCMIR3,x3xabcd3
              JR      C,ADDXXBK
INCTRCXX:    INC     @TRCXX
              JR      TRICONVXX

ADDXXBK:     CCF
              LD      lpcntr,x3xabcd
              ADC     RCMIR0,lpcntr
              LD      lpcntr,x3xabcd1
              ADC     RCMIR1,lpcntr
              LD      lpcntr,x3xabcd2
              ADC     RCMIR2,lpcntr
              LD      lpcntr,x3xabcd3
              ADC     RCMIR3,lpcntr

              DEC     RCPTR          ; next lower power of 3
              DEC     TRCXX          ; done with TRC00-TRC19 ?
              CP      TRCXX,#SYNC1   ; sync bit position?
              JR      nz,NXCP
              DEC     TRCXX          ;yes
NXCP:        CP      TRCXX,#1FH      ;no
              JR      nz,CALCTRNY

;*****
; Transmit initialization
;*****
;*****
; initialize RSFLAG
;*****
;
tm           RS232IP,#RS232IM        ;DATA IN LO?
JR           z,disrscall
ld           RSFLAG,#0FFh            ;set rs232 call enable flag

disrscall:   srp           #REGGRP40    ;set reg pntr
              LD          SYNC1,#02H    ;INITIALIZE SYNC1
              LD          acodeptr,#ACODE0BM-1 ;initialize
              LD          trcpttr,#SYNC0 ;for xmt
              LD          BITPTR,#0ffh
              LD          CODEPTR,#SYNC0
              LD          xmtreg,SYNC0
              LD          FRMCTRH,#02H   ;04H INIT FRAME COUNTER H
              LD          FRMCTRL,#0A0H  ;0BH INIT FRAME COUNTER L

```

A-11

# DCODETX.S

```

        clr     address                ;address for RS232 xfer
        LD      RS232DOCOUNT,#11D      ;turn off RS232 output
        LD      RS232DICOUNT,#0FFH     ;turn off RS232 input
                                           ;incoming data present
        LD      RSCOMMAND,#0FFH        ;turn off rs232 command
        clr     mtflag                 ;initialize mtflag
;*****
;      Wait for transmit INT
;*****
LOOP:    LD      IMR,#TIMER_ON_IMR      ;INT Mask enable
        EI                      ;enable INT
;*****RS-232 Routine*****
RSDATRDY: CP      RSCOMMAND,#0FFH ;RS232 DATA IN ?
        JR      Z,XMTMTL
        CP      mtflag,#0
        jr      z,RCVMTH
RCVMTH:  LD      mtempl,RS232DI ;input mtempl
        ld      RSCOMMAND,#0FFH
        clr     mtflag ;reset mtflag
        call    WRITEMEMORY ;write mtempl to EEprom
        call    READMEMORY ;read mtempl from EEprom
XMTMTH:  ld      RS232DO,mtempl ;rs232 echo back
        ld      RSSTART,#0FFH ;mtempl
        clr     RS232DOCOUNT
        ld      XMTFLAG,#0FFh ;set flag
        inc     address
        cp      address,#16D
        jr      nz,XMTMTL
        clr     address ;set address to 0
        jr      XMTMTL
RCVMTH:  ld      mtempl,RS232DI ;mtempl
        ld      RSCOMMAND,#0FFH
        ld      mtflag,#0FFH
XMTMTL:  cp      XMTFLAG,#0FFh ;ck for xmt first byte
        jr      nz,CKSWS
        cp      RS232DOCOUNT,#11D;test for output done
        jr      nz,CKSWS
        ld      RS232DO,mtempl ;echo back mtempl
        ld      RSSTART,#0FFH
        clr     XMTFLAG
;*****
CKSWS:   CP      FRMCTRH,#0 ;FRAME CTR = 0?
        JR      nz,LOOP
        cp      FRMCTRL,#0
        JR      nz,LOOP
SCHTOPP: STOP
;*****
;      TIMER 1 INTERRUPT ROUTINE
;*****
T1_INT:  CALL    CKB1
        EI                      ;enable interrupt
        CP      RSFLAG,#0FFh ;RS232 CALL ENABLE FLAG
        JR      nz,BEGIN

```

A-12

## DCODETX.S

```

call    RS232          ;RS232 I/O
push    RP             ;?

;*****INT pulse on P26*****
;
;      OR      P2,#01000000B    ;set P26 hi      *
;      NOP
;      AND     P2,#10111111b    ;set P26 lo      *
;*****
;*****FRAME 0 sync pulse on P26*****
;
;      CP      LPCNTR,#00H      ;testing frame sync pulse  *
;      JR      nz,NOSYNC        ;testing frame sync pulse  *
;      OR      P2,#01000000B    ;set frame sync pulse hi  *
;      JR      BEGIN
;NOSYNC:  AND     P2,#10111111b    ;set frame sync pulse lo  *
;*****
BEGIN:    INC      BITPTR          ;next bit
          CP      LPCNTR,#00      ;LPCNTR 0 ?
          JR      nz,NEXT
          CP      BITPTR,#00      ;BITPTR 0 ?
          JR      nz,NEXT
          SUB     FRMCTRL,#1      ;DECREMENT FRAME COUNTER
          SBC     FRMCTRH,#0
NEXT:     CALL    XMT             ;XMT next bit
          CP      LPCTR,#45       ;nibble 45?
          JR      nz,CKBP5
CKBP3:    CP      BITPTR,#1
          JR      z,BP00
          IRET
CKBP5:    CP      BITPTR,#03h
          JR      z,BP00
          IRET
BP00:     LD      BITPTR,#0FFH    ;reset bit pointer
          INC     LPCNTR          ;increment nibble pointer
CK2145:   CP      LPCNTR,#21      ;lpcntr>20?
          JR      mi,CK6790       ;no
LP46:     CP      LPCNTR,#46      ;yes,lpcntr<46
          JR      pl,CK6790
XMR00:    LD      xmtreg,#3       ;yes
          IRET
CK6790:   CP      LPCNTR,#67      ;no
          JR      mi,LP91
          CP      LPCNTR,#91
          JR      mi,XMR00
LP91:     CP      LPCNTR,#91      ;LPCNTR=91?
          JR      z,LPCTR00RET
LPCTR00RET:
          TM      LPCNTR,#00000001b ;LPCNTR bit0=0?
          JR      nz,INCACODE
          DEC     trcptr          ;no
          LD      CODEPTR,trcptr
          LD      xmtreg,@CODEPTR
          IRET
INCACODE: INC      acodeptr        ;yes
          LD      CODEPTR,acodeptr
          LD      xmtreg,@CODEPTR
          IRET

```

A-13

## DCODETX.S

```

LPCTR00:      clr      LPCNTR
              LD        TRCPTR,#SYNCO
              LD        acodeptr,#ACODE0BM-1
              LD        xmtreg,#SYNCO
              LD        CODEPTR,#SYNCO
              IRET

;*****
;      ADD TRINARY NUMBER TO ITSELF ROUTINE
;*****
AD3XX:        ADD      x3xabcd,x3xtmp ;add to itself
              ADC      x3xabcd1,x3xtmp1
              ADC      x3xabcd2,x3xtmp2
              ADC      x3xabcd3,x3xtmp3
              ret

;
XFER:         LD        x3xtmp,x3xabcd
              LD        x3xtmp1,x3xabcd1
              LD        x3xtmp2,x3xabcd2
              LD        x3xtmp3,x3xabcd3
              ret

;
ENTR3:        LD        x3xabcd,#03h
              clr      x3xabcd1
              clr      x3xabcd2
              clr      x3xabcd3
              ret

;
ENTR3A:       LD        x3xtmp,#03h
              clr      x3xtmp1
              clr      x3xtmp2
              clr      x3xtmp3
              ret

;*****
;      TRANSMIT ROUTINE
;*****
XMT:          CP        XMTREG,#3          ;BLANK TIME?
              JR        z,SBOLO            ;yes
              CP        XMTREG,#2          ;force trinary
              JR        ule,XMM
              ld        XMTREG,#2          ;TWO
XMM:          LD        XMTREG1,XMTREG      ;no,get xmt code
              COM       XMTREG1            ;compliment
              AND       XMTREG1,#00000011b ;mask 2 LSB
              CP        XMTREG1,BITPTR     ;compare bitptr to xmtreg
              JR        le,SBOHI
SBOLO:        AND       P0,#11111110b      ;set P00 lo
              RET
SBOHI:        OR        P0,#00000001b      ;set P00 hi
              RET

;*****
;      WRITE WORD TO MEMORY
;      ADDRESS IS SET IN REG ADDRESS
;      DATA IS IN REG MTEMPH AND MTEMPL
;      RETURN ADDRESS IS UNCHANGED
;*****
WRITEMEMORY:  push      RP                  ; SAVE THE RP

```

A-14

## DCODETX.S

```

    srp      #REGGRP40      ; set the register pointer

    call     STARTB         ; output the start bit
    ld       serial,#00110000B ; set byte to enable write
    call     SERIALOUT      ; output the byte
    and      csport,#csl    ; reset the chip select
    call     STARTB         ; output the start bit
    ld       serial,#01000000B ; set the byte for write
    or       serial,address ; or in the address
    call     SERIALOUT      ; output the byte
    ld       serial,mtemp    ; set the first byte to write

    call     SERIALOUT      ; output the byte
    ld       serial,mtempl   ; set the second byte to writ

e
    call     SERIALOUT      ; output the byte
    call     ENDWRITE       ; wait for the ready status
    call     STARTB         ; output the start bit
    clr      serial         ; set byte to disable write
    call     SERIALOUT      ; output the byte
    and      csport,#csl    ; reset the chip select
    pop      RP             ; reset the RP
    ret

```

```

;*****
; READ WORD FROM MEMORY
; ADDRESS IS SET IN REG ADDRESS
; DATA IS RETURNED IN REG MTEMPH AND MTEMPL
; ADDRESS IS UNCHANGED
;*****

```

```

READMEMORY:    CALL     CKB1
               push     RP
               srp      #REGGRP40      ; set the register pointer

               call     STARTB         ; output the start bit
               ld       serial,#10000000B ; preamble for read
               or       serial,address ; or in the address
               call     SERIALOUT      ; output the byte
               call     SERIALIN       ; read the first byte
               ld       mtemp,serial   ; save the value in mtemp
               call     SERIALIN       ; read teh second byte
               ld       mtempl,serial  ; save the value in mtempl
               and      csport,#csl    ; reset the chip select
               pop      RP
               ret

```

```

;*****
; START BIT FOR SERIAL NONVOL
; ALSO SETS DATA DIRECTION AND AND CS
;*****

```

```

STARTB:
    ld       P2M,#P2M_INIT ; set port 2 mode forcing output mode
data
    and      csport,#csl    ;
    and      clkport,#clockl ; start by clearing t

he bits
    and      dioport,#dol   ;
    or       csport,#csh    ; set the chip select
    or       dioport,#doh   ; set the data out high
    or       clkport,#clockh ; set the clock

```

A-15

## DCODETX.S

```

        and      clkport,#clockl      ; reset the clock low
        and      dioport,#dol         ; set the data low
        ret                               ; return

;*****
; END OF CODE WRITE
;*****
ENDWRITE:
        ld      P2M,#(P2M_INIT+1)      ; set port 2 mode forcing inp
ut mode data
        and      csport,#csl          ; reset the chip select
        nop                               ; delay
        or      csport,#csh          ; set the chip select
ENDWRITELOOP:
        WDT                               ; kick the dog
        cp      LPCNTRA,#1
        jr      nz,EWRLP
        call    CKB1
EWRLP:
        ld      temph,dioport          ; read the port
        and      temph,#doh            ; mask
        jr      z,ENDWRITELOOP        ; if the bit is low then loop till we
are done
        and      csport,#csl          ; reset the chip select
        ld      P2M,#P2M_INIT        ; set port 2 mode forcing output mode
        ret

;*****
; SERIAL OUT
; OUTPUT THE BYTE IN SERIAL
;*****
SERIALOUT:
        ld      P2M,#P2M_INIT        ; set port 2 mode forcing output mode
        data
        ld      templ,#8H            ; set the count for eight bit
S
SERIALOUTLOOP:
        rlc      serial                ; get the bit to output into
the carry
        jr      nc,ZEROOUT            ; output a zero if no carry
ONEOUT:
        or      dioport,#doh          ; set the data out high
        or      clkport,#clockh       ; set the clock high
        and      clkport,#clockl      ; reset the clock low
        and      dioport,#dol         ; reset the data out low
        djnz     templ,SERIALOUTLOOP  ; loop till done
        ret                          ; return
ZEROOUT:
        and      dioport,#dol         ; reset the data out low
        or      clkport,#clockh       ; set the clock high
        and      clkport,#clockl      ; reset the clock low
        and      dioport,#dol         ; reset the data out low
        djnz     templ,SERIALOUTLOOP  ; loop till done
        ret                          ; return

;*****
; SERIAL IN
; INPUTS A BYTE TO SERIAL
;*****
SERIALIN:
        ld      P2M,#(P2M_INIT+1)      ; set port 2 mode forcing inp

```

A-16

## DCODETX.S

```

ut mode data      ld      templ,#8H          ; set the count for eight bit
S
SERIALINLOOP:     or       clkport,#clockh     ; set the clock high
                  rcf      ; reset the carry flag
                  ld       temph,dioport       ; read the port
                  and      temph,#doh         ; mask out the bits
                  jr       z,DONTSET
                  scf      ; set the carry flag
DONTSET:          rlc      serial              ; get the bit into the byte
                  and      clkport,#clockl     ; reset the clock low
                  djnz     templ,SERIALINLOOP  ; loop till done
                  ret      ; return

```

-----

RS232 DATA ROUTINES

-----

enter rs232 start with word to output in rs232do

```

RS232OSTART:      clr      rsstart             ; one shot
                  ld       rs232odelay,#6d     ; set the time delay to 3. ms
                  clr      rs232docount        ; start with the counter at 0
                  and      RS232OP,#RS232OC    ; clear the output
                  jr       NORSOUT
RS232:            push     rp                  ; save the rp
                  srp      #REGGRP10          ; set the group pointer
                  cp       RSSTART,#OFFH      ; test for the start flag
                  jr       z,RS232OSTART
RS232OUTPUT:      cp       rs232docount,#11d   ; test for last
                  jr       nz,RS232R
                  or       RS232OP,#RS232OS   ; set the output idle
                  jr       NORSOUT
RS232R:           djnz     rs232odelay,NORSOUT ; cycle count time de
lay
                  inc      rs232docount        ; set the count for t
he next cycle
                  scf      ; set the carry flag
for stop bits
                  rrc      rs232do            ; get the data into t
he carry
                  jr       c,RS232SET         ; if the bit is high
then set
                  and      RS232OP,#RS232OC    ; clear the output
                  jr       SETTIME            ; find the delay time
RS232SET:         or       RS232OP,#RS232OS   ; set the output
SETTIME:          ld       rs232odelay,#6d    ; set the data output delay
                  tm       rs232docount,#00000001b ; test for odd words
                  jr       z,NORSOUT          ; if even done

```

A-17

## DCODETX.S

```

                ld      rs232odelay,#7d      ; set the delay to 7 for odd
                                                ; this gives 6.5 *.51
2mS
NORSOUT:
RS232INPUT:      cp      rs232dicount,#0FFH    ; test mode
                jr      nz,RECEIVING          ; if receiving then j
ump
                tm      RS232IP,#RS232IM        ; test the incoming d
ata for lo start bit
                jr      nz,NORSIN              ; if the line is stil
1 idle then skip
                clr     rs232dicount            ; start at 0
                ld      rs232idelay,#3         ; set the delay to mi
d
RECEIVING:       djnz   rs232idelay,NORSIN     ; skip till delay is
up
                inc     rs232dicount            ; bit counter
                cp      rs232dicount,#10d      ; test for last timeo
ut
                jr      z,DIEVEN
                tm      RS232IP,#RS232IM        ; test the incoming d
ata
                rcf     ; clear the carry
                jr      z,SKIPSETTING          ; if input bit not set skip s
etting carry
                scf     ; set the carry
SKIPSETTING:     rrc     rs232di               ; save the data into
the memory
                ld      rs232idelay,#6d        ; set the delay
                tm      rs232dicount,#00000001b ; test for odd
                jr      z,NORSIN                ; if even skip
                ld      rs232idelay,#7         ; set the delay
                jr      NORSIN
DIEVEN:          ld      rs232dicount,#0FFH    ; turn off the input
till next start
                ld      rscommand,rs232di      ; save the value
                clr     rsccount                ; clear the counter
NORSIN:          pop     rp                    ; return the rp
                ret
;*****CKB*****
CKB1:            WDT      ; HIT WDT
                tcm      P3,#S1                ;switch 1 pressed?
                jp      nz,CKB2
                clr     AC19                    ; ,#S1B39 yes
                ld      RCP,#RC10B             ;set rcptr s1
                RET
CKB2:            tcm      P3,#S2                ;no, switch 2 pressed?
                jp      nz,CKB3
                ld      AC19,#S2B39            ;yes
                ld      RCP,#RC20B             ;set rcptr s2
                RET
CKB3:            tcm      P3,#S3                ;no, switch 3 pressed?

```

A-1B



!

A-19

T0 SET TO 2uS clear each edge if timer extension times out then clear radio  
T1 set to 1uS for 256 uS roll to turn on the interrupts and to generate the 1 mS

Bit 35	Bit 37	Bit 39	ID_BIT	Type
0	0	Add In	0	Normal CMD
0	1	Add In	1	Touch code
0	2	Add In	2	Security
1	0	Add In	3	IR Protector
1	1	Key ID	4	Wall control
1	2	Key ID	5	Up Down CMD
2	0	Key ID	6	Up Down Stop
2	1	Don't learn	7	Open Door Indicator
2	2	Don't learn	8	Aux Function

#### NON-VOL MEMORY MAP

00	A1	RA1	RADIOP5
01	A1	RA1	RADIO1P5
02	A2	RC1	COUNTP5
03	A2	RC1	COUNT1P5
04	A3	RA2	
05	A3	RA2	
06	A4	RC2	
07	A4	RC2	
08	A5	RA3	
09	A5	RA3	
0A	A6	RC3	
0B	A6	RC3	
0C	A7	RA4	
0D	A7	RA4	
0E	A8	RC4	
0F	A8	RC4	
10	A9	RA5	
11	A9	RA5	
12	A10	RC5	
13	A10	RC5	
14	A11	RA6	
15	A11	RA6	
16	A12	RC6	
17	A12	RC6	
18	B	RA7	
19	B	RA7	
1A	C	RC7	
1B	C	RC7	
1C	CYCLE COUNTER 1ST 16 BITS		
1D	CYCLE COUNTER 2ND 16 BITS		
1E	VACATION FLAG		
1F	A MEMORY ADDRESS LAST WRITTEN		
	0XXXXXXX	ABC CODES	
	1XXXXXXX	D CODES	

20-2F OPERATION BACK TRACK

30-3F FORCE BACK TRACE

#### EQUATE STATEMENTS

check\_sum\_value .equ 0A2H  
TIMER\_0 .equ 10H  
TIMER\_0\_EN .equ 03H  
TIMER\_1\_EN .equ 0CH

P01M\_INIT .equ 00000100B ; set mode p00-p03 out  
P2M\_INIT .equ 00100100B  
P3M\_INIT .equ 00000011B ; set port3 p30-p33 ANALOG input  
P01S\_INIT .equ 00000000B  
P2S\_INIT .equ 00100110B  
P3S\_INIT .equ 00000000B

#### PERIODS

MONOPER .equ 38D ; MONOSTABLE PERIOD \*4mS  
RTOPERIOD .equ 130D ; period \*4mS => min 4\* period

#### INTERRUPTS

ALL\_ON\_IMR .equ 00111001b ; turn on int for radio  
RETURN\_IMR .equ 00111001b ; return on the IMR

#### Counter group

CounterGroup .equ 00 ; counter group  
LastM1Match .equ 05H ; last match 1 delay location  
LastMatch .equ 06H ; last matching code address  
LoopCount .equ 07H ; loop counter  
CounterA .equ 08H ; counter translation MSB  
CounterB .equ 09H  
CounterC .equ 0AH  
CounterD .equ 0BH ; counter translation LSB  
MirrorA .equ 0CH ; back translation MSB  
MirrorB .equ 0DH  
MirrorC .equ 0EH  
MirrorD .equ 0FH ; back translation LSB

```

loopcount      .equ    r7
countera       .equ    r8
counterb       .equ    r9
counterc       .equ    r10
counterd       .equ    r11
mirrora        .equ    r12
mirrorb        .equ    r13
mirrorc        .equ    r14
mirrord        .equ    r15

```

# ``` ; LEARN MODE SWITCHES AND ERASE ```

```

LearnModeGroup .equ    10H
SW_B           .equ    LearnModeGroup
CmdSwitch      .equ    LearnModeGroup+1 ; command switch
LearnDebounce  .equ    LearnModeGroup+2 ; learn switch debouncer
LearnTimer     .equ    LearnModeGroup+3 ; learn timer
SkipRadio      .equ    LearnModeGroup+4 ; flag to skip the radio read
ClearCount     .equ    LearnModeGroup+5
EraseTimer     .equ    LearnModeGroup+6 ; erase timer
BIT13          .equ    LearnModeGroup+7
BIT1P5        .equ    LearnModeGroup+8
ID_B           .equ    LearnModeGroup+9
LASTBIT        .equ    LearnModeGroup+10
PAST_MATCH     .equ    LearnModeGroup+11
Mono           .equ    LearnModeGroup+13
RadioTimeOut   .equ    LearnModeGroup+14 ; radio time out
SwitchSkip     .equ    LearnModeGroup+15

cmdswitch      .equ    r1
learndb        .equ    r2
learnt         .equ    r3
skipradio      .equ    r4
eraset         .equ    r6
rto            .equ    r14
mono           .equ    r13

```

# ``` ; LEARN EE GROUP FOR LOOPS ECT ```

```

LearnEeGroup   .equ    20H
TempH          .equ    LearnEeGroup
TempL          .equ    LearnEeGroup+1
Temp           .equ    LearnEeGroup+2
COUNT1P5H    .equ    LearnEeGroup+3 ; counter value memory
COUNT1P5L    .equ    LearnEeGroup+4 ; counter value memory
CMP            .equ    LearnEeGroup+5
MTempH        .equ    LearnEeGroup+6 ; memory temp
MTempL        .equ    LearnEeGroup+7 ; memory temp
MTemp         .equ    LearnEeGroup+8 ; memory temp
Serial         .equ    LearnEeGroup+9 ; serial data to and from nonvol memory
Addr ss       .equ    LearnEeGroup+10 ; address for the serial nonvol m mory
T0Ext         .equ    LearnEeGroup+11 ; timer 0 extend dec every T0 int
T4MS          .equ    LearnEeGroup+12 ; 4 mS counter

```

T125MS	.equ	LearnEeGroup+13	; 125mS counter
COUNTP5H	.equ	LearnEeGroup+14	; counter value memory
COUNTP5L	.equ	LearnEeGroup+15	; counter value memory
temph	.equ	r0	
templ	.equ	r1	
temp	.equ	r2	
cmp	.equ	r5	
mtemp	.equ	r6	; memory temp
mtempl	.equ	r7	; memory temp
mtemp	.equ	r8	; memory temp
serial	.equ	r9	; serial data to and from nonvol memory
address	.equ	r10	; address for the serial nonvol memory
t0ext	.equ	r11	; timer 0 extend dec every T0'int
t4ms	.equ	r12	; 4 mS counter
t125ms	.equ	r13	; 125mS counter

#### RADIO GROUP

RadioGroup	.equ	30H	
RTemp	.equ	RadioGroup	; radio temp storage
RTempH	.equ	RadioGroup+1	; radio temp storage high
RTempL	.equ	RadioGroup+2	; radio temp storage low
RTimeAH	.equ	RadioGroup+3	; radio active time high byte
RTimeAL	.equ	RadioGroup+4	; radio active time low byte
RTimeIH	.equ	RadioGroup+5	; radio inactive time high byte
RTimeIL	.equ	RadioGroup+6	; radio inactive time low byte
RadioP5H	.equ	RadioGroup+7	; .5 code storage
RadioP5L	.equ	RadioGroup+8	; .5 code storage
PointerH	.equ	RadioGroup+9	
PointerL	.equ	RadioGroup+10	
AddValueH	.equ	RadioGroup+11	
AddValueL	.equ	RadioGroup+12	
RadioC	.equ	RadioGroup+13	; radio word count
Radio1P5H	.equ	RadioGroup+14	; 1.5 code storage
Radio1P5L	.equ	RadioGroup+15	; 1.5 code storage
rtemp	.equ	r0	; radio temp storage
rtemph	.equ	r1	; radio temp storage high
rtempl	.equ	r2	; radio temp storage low
rtimeah	.equ	r3	; radio active time high byte
rtimeal	.equ	r4	; radio active time low byte
rtimeih	.equ	r5	; radio inactive time high byte
rtimeil	.equ	r6	; radio inactive time low byte
radiop5h	.equ	r7	; radio .5 code storage
radiop5l	.equ	r8	; radio .5 code storage
pointerh	.equ	r9	
pointerl	.equ	r10	
addvalueh	.equ	r11	
addvaluell	.equ	r12	
radioc	.equ	r13	; radio word count
radio1p5h	.qu	r14	; radio 1.5 code storage
radio1p5l	.equ	r15	; radio 1.5 code storage

```

.....
; Check sum group with past radio data
.....

```

```

CheckGroup      .equ    40H
check_sum       .equ    r0          ; check sum pointer
rom_data        .equ    r1
test_adr_hi     .equ    r2
test_adr_lo     .equ    r3
rflag          .equ    r4
test_adr        .equ    r2
pradioa         .equ    r6
pradiob         .equ    r7
pradioc         .equ    r8
pradiod         .equ    r9
pradioe         .equ    r10
pradiof         .equ    r11
pradiog         .equ    r12
pradioh         .equ    r13

Check_Sum       .equ    CheckGroup+0      ; check sum reg for por
Rom_Data        .equ    CheckGroup+1      ; data read
RFlag          .equ    CheckGroup+4      ; radio flags
RInFilter       .equ    CheckGroup+5      ; radio input filter
PRadioA         .equ    CheckGroup+6      ; past recieved value
PRadioB         .equ    CheckGroup+7      ; past recieved value
PRadioC         .equ    CheckGroup+8      ; past recieved value
PRadioD         .equ    CheckGroup+9      ; past recieved value
PRadioE         .equ    CheckGroup+0AH    ; past recieved value
PRadioF         .equ    CheckGroup+0BH    ; past recieved value
PRadioG         .equ    CheckGroup+0CH    ; past recieved value
PRadioH         .equ    CheckGroup+0DH    ; past recieved value

```

```

.....
; Timer group with rs232 data
.....

```

```

TimerGroup      .equ    50H
rs232do         .equ    r5
rs232di         .equ    r6
rscommand       .equ    r7
rs232dcount     .equ    r8
rs232dicount    .equ    r9
rs232odelay     .equ    r10
rs232idelay     .equ    r11
rs232ccount     .equ    r12
rs232page       .equ    r13
rsccount        .equ    r14
rsstart         .equ    r15

RADIO_CMD       .equ    TimerGroup+0H    ; radio command
TaskSwitch      .qu     TimerGroup+2H
SysDisable      .equ    TimerGroup+3H    ; system disable timer
ADD2            .equ    TimerGroup+4H

```



```

        .word    RadioPosInt          ;IRQ3, P3.2 p FOR EMULATION
        .word    TimerZeroInt         ; USE P3.0 FROM 28 PIN
        .word    TimerOneInt          ;IRQ4, T0
                                         ;IRQ5, T1

        .page
.org     000CH

```

# WATCHDOG INITIALIZATION

```

start:
START:    di                      ; turn off the interrupt for init
          WDH
          WDT                      ; kick the dog

```

## Internal RAM Test and Reset All RAM = mS

```

          srp    #0F0h             ; point to control group use stack
          ld     r15,#4             ; r15= pointer (minimum of RAM)

write_again:
          WDT                      ; KICK THE DOG
          ld     r14,#1

write_again1:
          ld     @r15,r14           ; write 1,2,4,8,10,20,40,80
          cp     r14,@r15           ; then compare
          jr     ne,system_error
          rl     r14
          jr     nc,write_again1
          clr    @r15               ; write RAM(r5)=0 to memory
          inc    r15
          cp     r15,#7FH
          jr     ult,write_again

```

## Checksum Test

```

ChecksumTest:
          srp    #CheckGroup
          ld     test_adr_hi,#07H
          ld     test_adr_lo,#0FFH ;maximum address=fffh

add_sum:
          WDT                      ; KICK THE DOG
          ldc    rom_data,@test_adr ;read ROM code one by one
          add    check_sum,rom_data ;add it to checksum register
          decw   test_adr           ;increment ROM address
          jr     nz,add_sum         ;address=0 ?
          cp     check_sum,#check_sum_value
          jr     system_ok
          jr     z,system_ok       ;check final checksum = 00 ?

system_error:
          and    P2,#11011101B    ; turn on the LED to indicate fault

```



```

        ld      P2M,#P2M_INIT                ; turn on the LED to indicate fault
        jr      system_error
system_ok:
        .byte   256-check_sum_value

        WDT                                     ; kick the dog

        srp     #LearnModeGroup              ; set the group

        ld      eraset,#0FFH                 ; set the erase timer
        ld      CmdSwitch,#0FFH             ; set the switch debouncer
        ld      learnt,#0FFH               ; set the learn timer
        ld      learndb,#0FFh              ; set the learn debounce
        ld      RSCommand,#0FFH            ; turn off the rs232 command
        ld      RS232DoCount,#11D          ; turn off the rs232 output

```

#### STACK INITIALIZATION

SetStack:

```

        clr     254
        ld      255,#STACKTOP               ; set the start of the stack

```

#### TIMER INITIALIZATION

```

        ld      PRE0,#00001001B            ; set the prescaler to / 2 for 8Mhz
        ld      PRE1,#00000111B            ; set the prescaler to / 1 for 8Mhz
        clr     T0                          ; set the counter to count FF through 0
        clr     T1                          ; set the counter to count FF through 0
        ld      TMR,#00001111B             ; turn on the timers and load

```

#### PORT INITIALIZATION

```

        ld      P0,#P01S_INIT              ; RESET all ports
        ld      P2,#P2S_INIT
        ld      P3,#P3S_INIT
        ld      P01M,#P01M_INIT            ; set mode
        ld      P3M,#P3M_INIT              ; set port3 p30-p33 input analog mode
        ld      P2M,#P2M_INIT+1           ; set port 2 mode

```

#### MEMORY INITIALIZATION

```

        ld      Address,#3EH               ; set non vol address to UNUSED
        call    ReadMemory                 ; read the value to INIT

```

```

.....
; INTERRUPT INITIALIZATION
;
.....

```

SetInterrupts:

```

ld    IPR,#00000001B    ; set the priority to timer
ld    IMR,#ALL_ON_IMR   ; turn on the interrupt
clr    IRQ               ; CLEAR IRQ'S

```

```

.....
; MAIN LOOP
;
.....

```

MainLoop:

```

    ei                    ; enable interrupt
    and    P2,#01111111b ; turn off the flag
    wdt     ; kick the dog
    ld    P01M,#P01M_INIT ; set mode
    ld    P3M,#P3M_INIT   ; set port3 p30-p33 input analog mode
    ld    P2M,#P2M_INIT+1 ; set port 2 mode

    call    LEARN          ; do the learn switch

TestRS232:
    srp    #TimerGroup    ;
    cp     rsstart,#0FFH   ; test for starting a transmission
    jr     z,skipsr232     ; if starting a trans skip
    cp     rscommand,#0FFH ; test for the off mode
    jr     z,skipsr232
    cp     rs232docount,#11d ; test for output done
    jr     nz,skipsr232    ; if not the skip
    cp     rscommand,#30H   ; test for switch data
    jr     nz,TEST34
    clr     rs232do         ; clear the data

    cp     LearnDebounce,#0FFH ; test switch one
    jr     nz,SW1OUT
    or     rs232do,#00000001B ; set the marking bit

;SW1OUT:
    cp     CmdSwitch,#0FFH   ; test switch 2
    jr     nz,SW2OUT
    or     rs232do,#00000010B ; set the marking bit

;SW2OUT:
    cp     LearnTimer,#0FFH   ; test for learn 1
    jr     nz,L1OUT
    or     rs232do,#00001000B ; set the marking bit

;L1OUT:
    jr     VacSwOpen

TEST34:
    cp     rscommand,#34H   ; test for page 0
    jr     nz,TEST35
    ld     rs232page,#00H
    jr     RS232PageOUT

TEST35:
    cp     rscommand,#35H   ; test for page 1
    jr     nz,TEST38

```

	ld	rs232page,#10H	
RS232PageOUT:	ld	SkipRadio,#0FFH	; set the skip radio flag
	dec	SwitchSkip	; turn off the switch testing for port
			; direction control
	ld	Address,rscount	; find the address
	rcf		
	rrc	Address	
	or	Address,rs232page	
	call	ReadMemory	; read the data
	ld	rs232do,MTempH	
	tm	rscount,#01H	; test which byte
	jr	z,RPBYTE	
	ld	rs232do,MTempL	
RPBYTE:	cp	rscount,#1FH	; test for the end
	jp	nz,STARTOUT	
LASTRPM:	clr	rscount	; reset the counter
VacSwOpen:			
	dec	rsstart	; set the start flag
	ld	rscommand,#0FFH	; turn off command
			; return
skips232:			
	jp	SKIPRS232	
TEST38:			
	cp	rscommand,#38H	; test memory
	jr	nz,SKIPRS232	
	ld	rs232do,#0FFH	; flag set to error to start
	srp	#LearnEeGroup	
	dec	SwitchSkip	; skip testing the switches
	ld	SkipRadio,#0FFH	; set the skip radio flag
	ld	mtempH,#0FFH	; set the data to write
	call	WRITEALL	; write all the words
	call	TESTALL	; test all memory
	ld	mtempH,#000H	; set the data to write
	call	WRITEALL	; write all memory
	call	TESTALL	; test for the data retention
CLEARALL:			
	call	CLEARCODES	; reset the memory for code
	clr	RS232DO	; flag all ok
MEMORYERROR:			
	ld	RSCommand,#0FFH	; turn off command
STARTOUT:			
	inc	rscount	; set to the next address
	dec	RSStart	; set the start flag
SKIPRS232:			
	clr	SwitchSkip	; clear the skip switches flag
	clr	SkipRadio	; clear the skip radio flag
	srp	#LearnModeGroup	
SINGLE:			
	cp	mono,#MONOPER	; test for the period

```

        jr      ult,TESTCONS          ; if not then test constant output
        and     P2,#11110111b        ; clear the output
        ld      mono,#0FFH
TESTCONS:
        di
        cp     rto,#RTOPERIOD        ; test for the timeout
        jr      ult,SIGDONE
TurnOffOutput:
        and     P2,#11101111b        ; clear the output
        ld      rto,#0FFH
SIGDONE:
TOGGLE:
        jp      MainLoop              ; loop forever

WRITEALL:
        ld      mtempl,mtempH
        ld      TestVal,mtempH
        clr     address                ; start at address 00
WRITELOOP1:
        WDT
        call    WRITEMEMORY
        inc     address                ; do the next address
        cp     address,#40H           ; test for the last address
        jr      nz,WRITELOOP1

TESTALL:
        clr     address                ; start at address 0
READLOOP1:
        WDT
        call    ReadMemory             ; read the data
        cp     mtempH,TestVal          ; test the value
        jp     nz,MEMORYERROR          ; if error mark
        cp     mtempl,TestVal          ; test the value
        jp     nz,MEMORYERROR          ; if error mark
        inc     address                ; set the next address
        cp     address,#40H           ; test for the last address
        jr      nz,READLOOP1
        ret

```

```

.....
; Timer 0 interrupt
.....

```

TimerZeroInt:

```

        cp     T0Ext,#00              ; test for the roll
        jr     z,ClearRadioTimeout    ; if at the roll time out
        dec     T0Ext                 ; decrement the time extension
        iret
ClearRadioTimeout:
        call    ClearCounter           ; clear the counter
        push    RP                    ; for the Clear radio code segment
        jp      ClearRadio             ; clear the radio data

```

```

.....
; Radio interrupt from a edge of the radio signal
.....

```

```

RadioNegInt:
    and    IMR,#11111110b    ; turn off the interrupt for 256uS
    ld     RTemp,#00000001B  ; mark which edge
    jr     RadioEdge

```

```

RadioPosInt:
    and    IMR,#11110111b    ; turn off the interrupt for 256uS
    ld     RTemp,#00000000B  ; mark which edge
    jr     RadioEdge

```

RadioEdge:

```

    push   RP                ; save the reg pair
    srp    #RadioGroup       ; set the register pointer
    ld     rtemp,T0Ext        ; read the upper byte
    ld     rtempl,T0          ; read the lower byte
    tm     IRQ,#00010000b     ; test for a pending timer interrupt
    jr     z,RIncDone         ; done
    tm     rtempl,#10000000b  ; test for the rollover
    jr     z,RIncDone         ; if not the rolled value skip inc
    dec    rtemp              ; increase the timer msb

```

```

RIncDone:
    call   ClearCounter      ; clear the counter

```

```

RTimeOk:
    com    rtemp              ; flip to find the period
    com    rtempl

```

```

RTimeDone:
    cp     rtemp,#0           ; test the port for the edge
    jr     z,ActiveTime       ; if it was the active time then branch

```

```

InActiveTime:
    cp     RInFilter,#0FFH    ; test for active last time
    jr     z,GoInActive       ; if so continue
    jr     RADIO_EXIT         ; if not the return

```

```

GoInActive:
    clr    RInFilter          ; set flag to inactive
    ld     rtimeih,rtemp      ; transfer the period to inactive
    ld     rtimeil,rtempl
    jr     RADIO_EXIT         ; return

```

```

ClearCounter:
    ld     TMR,#00001000b     ; turn off timer 0
    ld     TMR,#00001001b     ; load t0
    ld     TMR,#00001000b
    ld     TMR,#00001010b     ; restart the timer
    ld     T0Ext,#0FFH        ; reset the timer
    and    IRQ,#11100110b     ; turn off pending int
    ret

```

```

ActiveTime:
    cp     RInFilter,#00H     ; test for active last time
    jr     z,GoActive         ; if so continue
    jr     RADIO_EXIT         ; if not the return

```

GoActive:

```

ld      RInFilter,#0FFH
ld      rtimeah,rtempH
ld      rtimeal,rtempL
GotBothEdges:
ei
cp      radioc,#0
jr      nz,INSIG
inc     radioc
cp      rtimeih,#30h
jr      ult,ClearJump
cp      rtimeah,#00h
jr      nz,SyncOk
cp      rtimeal,#80H
jr      ult,ClearJump
SyncOk:
cp      rtimeah,#9h
jr      uge,ClearJump

SETP5:
cp      rtimeah,#02h
jr      uge,O1P5MSFLAG
P5MSFLAG:
or      RFlag,#01000000b
clr     radiop5h
clr     radiop5l
clr     COUNTP5H
clr     COUNTP5L
jr      DONESETP5
O1P5MSFLAG:
and     RFlag,#10111111b
clr     radio1p5h
clr     radio1p5l
clr     COUNT1P5H
clr     COUNT1P5L
DONESETP5:
RADIO_EXIT:
pop     rp
iret

ClearJump:
;
or      P2,#10000000b
jp      ClearRadio
INSIG:
cp      rtimeih,#0AH
jr      uge,ClearJump
cp      rtimeih,#00h
jr      nz,ISigOk
cp      rtimeil,#080h
jr      ult,ClearJump
ISigOk:
cp      rtimeah,#0AH
jr      uge,ClearJump
cp      rtimeah,#00h

```

```

; transfer the period to active

```

```

; enable the interrupts
; test for the blank timing
; if not then in the middle of signal
; set the counter to the next number
; test for the min 24.5 mS
; if not then clear the radio
; test first the min sync
; first byte 00 if not great enough
; test for 256uS min
; if less then clear the radio

```

```

; test for the max time 4.6mS
; if not clear

```

```

; test for 1.5 vs .5
; set the 1.5 flag

```

```

; set the 0.5ms memory flag
; clear the memory
; clear the memory
; do the 2X

```

```

; set the 1.5ms memory flag
; clear the memory
; clear the memory

```

```

; done return

```

```

; turn of the flag bit for clear radio
; clear the radio signal

```

```

; test for the max width 5.16
; if too wide clear
; test for the min width
; if greater then 0 then signal ok
; test for 256us min
; if not then clear the radio

```

```

; test for the max width
; if too wide clear
; if greater then 0 then signal ok

```

	jr	nz,ASigOk	; if too narrow clear
	cp	rtimeal,#080h	; test for 256us min
ASigOk:	jr	ult,ClearJump	; if not then clear the radio
	sub	rtimeal,RTimeLL	; find the difference
	sbc	rtimeah,rtimeih	
	tm	rtimeah,#10000000b	; find out if neg
	jr	nz,NEGDIFF2	; use 1 for ABC or D
POSDIFF2:	jr	POSDIFF2	
	cp	rtimeah,#01H	; test for 1.5/1
	jr	ult,O1PMS	; mark as a 1
	jr	O1P5MS	
NEGDIFF2:			
	com	rtimeah	; invert
	cp	rtimeah,#01H	; test for 1/.5
	jr	ult,O1PMSC	; mark as a .5
	jr	P5MSC	
O1P5MS:			
	ld	BIT1P5,#2h	; set the value
	jr	GOTB1P5	
O1PMSC:			
	com	rtimeah	; invert
O1PMS:			
	ld	BIT1P5,#1h	; set the value
	jr	GOTB1P5	
P5MSC:			
	com	rtimeah	; invert
	ld	BIT1P5,#0h	; set the value
GOTB1P5:			
	clr	rtimeah	; clear the time
	clr	rtimeal	
	clr	rtimeih	
	clr	rtimeil	
	ei		; enable interrupts
ADDB1P5:			
	tm	RFlag,#01000000b	; test for radio p5/ 1p5
	jr	nz,RCP5INC	
RC1P5INC:			
	tm	radioc,#00000001b	; test for even odd number
	jr	z,COUNT1P5INC	; if odd number counter
Radio1P5INC:			; else radio
	cp	radioc,#15D	; test the radio counter for the specials
	jr	uge,SPECIAL_BITS	; save the special bits separate
Radio1P5R:			
	ld	pointerh,#Radio1P5H	; get the pointer
	ld	pointerl,#Radio1P5L	
	jr	AddAll	
SPECIAL_BITS:			
	cp	radioc,#15d	; test for the first special
	jr	nz,SKIP_ID_ZERO	; if not then skip zeroing
	clr	ID_B	; else clear the id bits

```

SKIP_ID_ZERO:
    cp    radioc,#19d          ; test for the switch id
    jr    z,SWITCHID          ; if so then branch

    ld    rtemp,ID_B          ; save the special bit
    add   ID_B,rtemp          ; *3
    add   ID_B,rtemp          ; *3
    add   ID_B,BIT1P5         ; add in the new value
    jr    Radio1P5R

SWITCHID:
    ld    SW_B,BIT1P5         ; save the switch ID
    cp    ID_B,#03d          ; test for the add in values
    jr    c,Radio1P5R         ; add in if 3 <
    clr   BIT1P5              ; else dont add in
    jr    Radio1P5R

RCP5INC:
    tm    radioc,#00000001b   ; test for even odd number
    jr    z,COUNTP5INC       ; if odd number counter

RadioP5INC:
    ld    pointerh,#RadioP5H   ; else radio
    ld    pointerl,#RadioP5L   ; get the pointer
    jr    AddAll

COUNT1P5INC:
    ld    pointerh,#COUNT1P5H ; get the pointer
    ld    pointerl,#COUNT1P5L
    jr    AddAll

COUNTP5INC:
    ld    pointerh,#COUNTP5H  ; get the pointers
    ld    pointerl,#COUNTP5L
    jr    AddAll

AddAll:
    ld    rtemp,@pointerh      ; get the value
    ld    rtempl,@pointerl     ; get the value
    ld    addvalueh,@pointerh
    ld    addvaluel,@pointerl

    add   addvalueh,rtempl      ; add x2
    adc   addvalueh,rtemp      ; add x3
    add   addvalueh,rtemp
    add   addvalueh,BIT1P5     ; add in new number
    adc   addvalueh,#00h
    ld    @pointerh,addvalueh  ; save the value
    ld    @pointerl,addvalueh

ALLADDED:
    inc   radioc              ; increase the counter

TWENTY?:
    and   RFlag,#11011111B    ; clear the bit for 10 bits
    cp    radioc,#21D          ; test for 20
    jp    nz,RRETURN          ; if not then return
    tm    RFlag,#00010000B     ; test flag 20 bit code

```



```

FIRST20:    jr      nz,KNOWCODE          ; if the second 20 bits received
            or      RFlag,#00010000B    ; set the flag
            clr     radioc                ; clear the radio counter
            jp      RRETURN              ; return
GOT20CODE:  cp      ID_B,#07d            ; test for the don't use ones
            jp      uge,ClearRadio        ; clear don't use
            cp      ID_B,#04d            ; test for the don't add in ones
            jr      uge,KNOWCODE          ; if so then don't add in
            add     COUNT1P5L,SW_B        ; add in switch id
            adc     COUNT1P5H,#00h

```

KNOWCODE:

.....  
Translate the counter back to normal

start	CounterA	CounterB	CounterC	CounterD
	00	00	Count1P5H	Count1P5L
	MirrorA	MirrorB	MirrorC	MirrorD
	00	00	CountP5H	CountP5L

.....

```

srp        #CounterGroup                ; set the group
clr        countera                      ; clear the counter Msb value
clr        counterb
ld         counterc,COUNT1P5H            ; Set the value to count1p5
ld         counterd,COUNT1P5L
clr        mirrora                      ; Set the mirror (temp reg for now)
clr        mirrorb                      ; to countp5
ld         mirrorc,COUNTP5H
ld         mirrored,COUNTP5L
call       AddMirrorToCounter            ; find countp5 * 3^10 + count1p5
ld         loopcount,#3
call       RotateMirrorAdd
ld         loopcount,#2
call       RotateMirrorAdd
ld         loopcount,#2
call       RotateMirrorAdd
ld         loopcount,#2
call       RotateMirrorAdd
ld         loopcount,#1
call       RotateMirrorAdd
ld         loopcount,#3
call       RotateMirrorAdd
ld         loopcount,#1
call       RotateMirrorAdd
ld         loopcount,#1
call       RotateMirrorAdd

```

MirrorTheCounter:

```

call       MirrorCounter                ; mirror the counter

```

CounterCorrected:

```

cp         SkipRadio,#0FFH              ; test for the skip radio flag
jp         z,ClearRadio                 ; if active do not test the cpde
cp         LearnTimer,#0FFH             ; test for in learn mode

```

```

                                jp      z,TESTCODE                ; if not in learn the test the code
STORECODE:
DCODESTORE:
                                cp      PRadioA,radio1p5h        ; test all 8 memorys for a match
                                jr      nz,PP_NOT_M_D            ; if no match skip
                                cp      PRadioB,radio1p5l        ; test all 8 memorys for a match
                                jr      nz,PP_NOT_M_D            ; if no match skip
                                cp      PRadioC,radio1p5h        ; test all 8 memorys for a match
                                jr      nz,PP_NOT_M_D            ; if no match skip
                                cp      PRadioD,radio1p5l        ; test all 8 memorys for a match
                                jr      nz,PP_NOT_M_D            ; if no match skip
                                cp      PRadioE,MirrorA          ; test all 8 memorys for a match
                                jr      nz,PP_NOT_M_D            ; if no match skip
                                cp      PRadioF,MirrorB          ; test all 8 memorys for a match
                                jr      nz,PP_NOT_M_D            ; if no match skip
                                cp      PRadioG,MirrorC          ; test all 8 memorys for a match
                                jr      nz,PP_NOT_M_D            ; if no match skip
                                cp      PRadioH,MirrorD          ; test all 8 memorys for a match
                                jr      nz,PP_NOT_M_D            ; if no match skip
MatchedForStore:
                                srp      #LearnEeGroup
                                call     TESTMATCH                ; test for a matching code
                                cp      address,#0FFH            ; test for a match
                                jr      nz,WRITEAGAIN            ; if so store AGAIN for counter
                                ld      address,#1FH              ; set the address
                                call     ReadMemory               ; read the value
                                add      mtempH,#4d              ; find the next address
                                cp      mtempH,#1CH              ; test for out of range
                                jr      ult,GOTDADDRESS
                                clr      mtempH
GOTDADDRESS:
                                ld      mtempL,mtempH
                                ld      address,#1FH              ; store the new address
                                call     WRITEMEMORY
                                ld      address,mtempH
                                call     WRITE_D_CODE             ; set the code address to write
                                jr      NOWRITESTORE              ; output the D code
                                ; reset the learn mode
WRITEAGAIN:
                                call     WRITE_D_CODE              ; output the D code
NOWRITESTORE:
                                or      P2,#00000010B           ; turn off the LED for flashing
                                ld      LearnTimer,#0FFH         ; turn off the learn mode
                                clr      RadioTimeOut            ; disable command from learn
                                jr      ClearRadio                ; set for the next code
PP_NOT_M_D:
                                ld      PRadioA,radio1p5h        ; save the present into the past
                                ld      PRadioB,radio1p5l        ; save the present into the past
                                ld      PRadioC,radio1p5h        ; save the present into the past
                                ld      PRadioD,radio1p5l        ; save the present into the past
                                ld      PRadioE,MirrorA          ; save the present into the past
                                ld      PRadioF,MirrorB          ; transfer the value

```

```

ld    PRadioG,MirrorC
ld    PRadioH,MirrorD
; reset radio

;.....
; Clear interrupt
;.....
ClearRadio:
tm    RFlag,#00000001B    ; test for receiving without error
jr    z,SKIPiRTO          ; if flag not set then donot clear timer
clr    RadioTimeOut       ; clear radio timer

SKIPiRTO:
clr    RadioC             ; clear the radio counter
clr    RFlag              ; clear the radio flags

RRETURN:
pop    RP                 ; reset the RP
iret                    ; return

;.....
; rotate mirror LoopCount * 2 then add
;.....
RotateMirrorAdd:

rcf                    ; clear the carry
rlc    mirrord
rlc    mirrorc
rlc    mirrorb
rlc    mirrora
djnz   loopcount,RotateMirrorAdd ; loop till done
;.....
; Add mirror to counter
;.....
AddMirrorToCounter:
add    counterd,mirrord
adc    counterc,mirrorc
adc    counterb,mirrorb
adc    countera,mirrora
ret

;.....
; Add mirror to counter
;.....
MirrorCounter:
ld    loopcount,#32d    ; set the number of bits

MirrorLoop:
rrc    countera
rrc    counterb
rrc    counterc
rrc    counterd
rlc    mirrord
rlc    mirrorc
rlc    mirrorb
rlc    mirrora
djnz   loopcount,MirrorLoop ; loop for all the bits
ret

```

.....  
 Test the radio code for matching  
 .....

TESTCODE:

```

and    P2,#11111101B      ; turn on the LED for flashing
srp    #LearnEeGroup
call   TESTMATCH          ; test the code for a match
or     P2,#00000010B      ; turn off the LED for flashing
cp     Address,#0FFH      ; test for no match
jp     z,TEST_TC_SEC      ; if no match try touchcode and sec

```

D\_CODE\_MATCH:

```

cp     RadioTimeOut,#0FFH ; test for the timeout
jr     z,NewCode          ; if timer inactive then look for a new
cp     LastM1Match,Address ; test for the same address as the past
jr     nz,NewCode         ; if not then test for a new code
clr    RadioTimeOut       ; reclear the timer
jp     ClearRadio         ; and update the past

```

NewCode:

```

srp    #CheckGroup        ; set the rp
call   TESTCOUNTER        ; test the counter for in range
cp     CMP,#00             ; test for a matching value
jp     z,ClearRadio        ; if the same then clear the radio
cp     CMP,#0AAH           ; test for counter in range
jr     z,GOT_D_CMD        ; got a command save radio counter
cp     CMP,#07FH           ; test for outside of - window
jr     z,UPDATE_PAST      ; if so skip resync
cp     PAST_MATCH,Address  ; test for the same address as the past
jr     nz,UPDATE_PAST     ; if not then update the past value
ld     pradioa,MirrorA    ; transfer the value
ld     pradiob,MirrorB
ld     pradioc,MirrorC
ld     pradiod,MirrorD
sub    pradiod,pradioh
sbc    pradioc,pradiog
sbc    pradiob,pradiof
sbc    pradioa,pradioe
cp     pradioa,#00         ; find the difference
jr     nz,UPDATE_PAST     ; test for less then 4 away
cp     pradiob,#00         ; if not then update the past
jr     nz,UPDATE_PAST     ; if not then update the past
cp     pradioc,#00         ; if not then update the past
jr     nz,UPDATE_PAST     ; test for the zero case
cp     pradiod,#00
jr     z,UPDATE_PAST
cp     pradiod,#04d
jr     ugt,UPDATE_PAST

```

GOT\_D\_CMD:

```

jr     ugt,UPDATE_PAST    ; if not then update the past
call   STORE_D_COUNTER    ; save the new counter value

```

D\_RADIO\_COMMAND

```

cp     SysDisable,#32d    ; test for 4 seconds
jr     ult,TEST_TC_SEC   ; if not test tc and sec

cp     RadioTimeOut,#RTOPERIOD ; test for first reception

```

```

        jr      ult,NOTP3A          ; if second reception skip t and mono
        clr     Mono                ; clear the monostable
        or      P2,#00011000B      ; turn on the constant
        xor     P2,#01000000B      ; toggle the T output
NOTP3A:  clr     RadioTimeOut        ; clear the timer
NOTP3:
NOTP3S:
        jr      TEST_TC_SEC        ; test tc and sec

NOTNEWMATCH:
        ld      LearnTimer,#0FFH   ; set the learn timer "turn off"
        jp      ClearRadio         ; clear the radio

UPDATE_PAST:
        ld      PAST_MATCH,Address ; save the past address
        ld      pradioe,MirrorA    ; transfer the value
        ld      pradiof,MirrorB
        ld      pradiog,MirrorC
        ld      pradioh,MirrorD
        jp      ClearRadio         ; reset the radio

```

.....

We know the code does not match but if it was our touch code  
or security transmitter update the counter

.....

```

TEST_TC_SEC:
        srp     #LearnEeGroup
        cp      ID_B,#1d           ; test for the touch code
        jr      z,TC_SEC           ; jump if so
        cp      ID_B,#2d           ; test for the security transmitter
        jr      z,TC_SEC           ; jump if so
        jp      ClearRadio

TC_SEC:
        ld      address,#01d       ; set the start addresss for the fixed

NEXT_D:
        call    ReadMemory          ; read the word at this address
        cp      mtempH,Radio1P5H    ; test for the match
        jr      nz,NO_TC_MATCH      ; if not matching do the next address
        cp      mtempL,Radio1P5L    ; test for the match
        jr      nz,NO_TC_MATCH      ; if not matching do the next address
        dec     address             ; reset the address

MatchedCheckCounter:
        call    TESTCOUNTER         ; test the counter for in range
        cp      CMP,#0AAH           ; test for within range
        jr      nz,SkipStoreCounter ; if not kip storing the counter

TC_SEC_Store:
        call    STORE_D_COUNTER     ; save the new counter

SkipStoreCounter:
        inc     address

NO_TC_MATCH:
        add     address,#4d         ; set the address to the next code
        cp      address,#1CH        ; test for the last address

```

```

        jr      ult,NEXT_D          ; if not the last address then try again

GOTNO_TC_MATCH:
        jp      ClearRadio

```

```

.....
Test the radio code counter and compares
CMP
00 => counter the same
FF => counter out of range
AA => counter in range
7F => counter within - window no resync
Address for test in address
.....

```

#### TESTCOUNTER:

```

        push    RP                  ; save the RP
        srp     #CheckGroup        ; set the rp
        inc     Address             ; set the address to the 2x position for
        inc     Address
        call    ReadMemory          ; read the value
        ld      pradioa,MTempH      ; temp storage
        ld      pradiob,MTempL
        inc     Address
        call    ReadMemory          ; read the value
        sub     Address,#3d         ; reset the address
        ld      pradioc,MTempH      ; temp storage
        ld      pradiod,MTempL
        cp      MirrorA,pradioa     ; test first for the match
        jr      nz,NM_COUNTER       ; if not then test count position
        cp      MirrorB,pradiob
        jr      nz,NM_COUNTER       ; if not then test count position
        cp      MirrorC,pradioc
        jr      nz,NM_COUNTER       ; if not then test count position
        cp      MirrorD,pradiod
        jr      nz,NM_COUNTER       ; if not then test count position
        ld      CMP,#00h            ; flag the match

```

#### CounterRet:

```

        pop     RP
        ret

```

#### NM\_COUNTER:

```

        cp      pradioa,#0FFH       ; test for the roll over
        jr      nz,NORMALN          ; if not test normally
        cp      pradiob,#0FFH       ; test for the roll over
        jr      nz,NORMALN          ; if not test normally
        cp      MirrorA,#0H         ; test for the rollover
        jr      nz,NORMALN          ; if not test normally
        cp      MirrorB,#0H         ; test for the rollover
        jr      nz,NORMALN          ; if not test normally
        ; at roll com past add pres

        call    Complement
        add     pradiod,MirrorD      ; add the 2
        adc     pradioc,MirrorC
        adc     pradiob,MirrorB
        adc     pradioa,MirrorA

```

```

COUNTOUT:      cp      pradioc,#12d      ; window 3072 or 1024 activations
                 jr      ule,COUNTOK

```

```

                 call     Complement      ; find the - difference
                 cp      pradioa,#00     ; test for within 00000400H
                 jr      nz,OutOfWindow
                 cp      pradiob,#00
                 jr      nz,OutOfWindow
                 cp      pradioc,#00000100B
                 jr      ugt,OutOfWindow
                 ld      CMP,#7FH        ; mark the -window function
                 jr      CounterRet      ; return

```

OutOfWindow:

```

                 ld      CMP,#0FFH      ; set the bad count flag
                 jr      CounterRet      ; return

```

COUNTOK:

```

                 ld      CMP,#0AAH      ; set the count flag ok
                 jr      CounterRet      ; return

```

NORMALN:

```

                 sub      pradiod,MirrorD      ; subtrace to find difference
                 sbc      pradioc,MirrorC
                 sbc      pradiob,MirrorB
                 sbc      pradioa,MirrorA
                 call     Complement
                 cp      pradioa,#00        ; make positive
                 jr      nz,COUNTOUT        ; test for to large
                 cp      pradiob,#00        ; if so out of window
                 jr      nz,COUNTOUT        ; test for to large
                 cp      pradioc,#11D       ; if so out of window
                 jr      ule,COUNTOK        ; window for 1024
                 jr      COUNTOUT

```

Complement:

```

                 com      pradiod          ; Complement the temp reg
                 com      pradioc
                 com      pradiob
                 com      pradioa
                 ret

```

```

.....
; TESTMATCH TEST THE NON ROLLING PART OF ANY CODE IF THERE
; IS A MATCH RETURNS THE ADDRESS ELSE RETURNS FF
;
.....

```

TESTMATCH:

TEST\_D\_CODES:

```

                 clr      address          ; start at address 0

```

NEXT\_D\_CODE:

```

                 call     ReadMemory      ; read the word at this address
                 cp      mtempH,RadioP5H ; test for the match
                 jr      nz,NO_D_MATCH    ; if not matching then do n xt address
                 cp      mtempL,RadioP5L ; test for the match
                 jr      nz,NO_D_MATCH    ; if not matching then do n xt address

```

```

        inc     address                ; set the second half of the code
        call    ReadMemory             ; read the word at this address
        cp      mtempH, Radio1P5H      ; test for the match
        jr      nz, NO_D_MATCH2        ; if not matching do the next address
        cp      mtempL, Radio1P5L      ; test for the match
        jr      nz, NO_D_MATCH2        ; if not matching do the next address
        dec     address                ; reset the address
        jr      TMEXIT                 ; return with the address of the match

NO_D_MATCH:
        inc     address                ; set the address to the next code
NO_D_MATCH2:
        add     address, #3d           ; set the address to the next code
        cp      address, #1CH          ; test for the last address
        jr      ult, NEXT_D_CODE       ; if not the last address then try again

GOTNO_D_MATCH:
        ld      address, #0FFH         ; set the no match flag
        ret

TMEXIT:
        ld      LastM1Match, LastMatch ; delay line
        ld      LastMatch, address     ; save the address for radio timeout
        ret

```

```

.....
; LEARN DEBOUNCES THE LEARN SWITCH 80mS
; TIMES OUT THE LEARN MODE 30 SECONDS
; DEBOUNCES THE LEARN SWITCH FOR ERASE 6 SECONDS
.....

```

```

LEARN:
        srp     #LearnModeGroup        ; set the group
        cp      cmdswitch, #236D       ; test for the debouncer release
        jr      nz, ReleaseDone        ; if not then test for set
        clr     cmdswitch              ; clear the debouncer

ReleaseDone:
        cp      cmdswitch, #20D        ; test for switch 2 set
        jr      UGT, CLEARRA           ;

multi2:
        cp      cmdswitch, #20D        ; test for switch 2 set
        jr      nz, TESTLEARN         ; if not then test learn

SW2isSET:
        ld      cmdswitch, #0FFH       ; set the debouncer

CMDSW:
        clr     mono                   ; clear the timer
        xor     P2, #01000000B         ; toggle
        or      P2, #00011000B         ; set

CLEARRA:
        clr     rto                   ;

TESTLEARN:
        cp      learndb, #236D         ; test for the debounced release
        jr      nz, LEARNNOTRELEASED   ; if not released then jump

```



```

        clr     learndb                ; clear the debouncer
        ret                                ; return

LEARNNOTRELEASED:
        cp     learnt,#0FFH           ; test for learn mode
        jr     nz,INLEARN              ; if in learn jump
        cp     learndb,#20D           ; test for debounce period
        jr     nz,ERASETEST           ; if not then test the erase period

SETLEARN:
        clr     learnt                ; clear the learn timer
        ld     learndb,#0FFH          ; set the debouncer
        and     P2,#11111101b        ; turn on the led

ERASETEST:
        cp     learndb,#0FFH          ; test for learn button active
        jr     nz,ERASERELEASE        ; if button released set the erase timer
        cp     eraset,#0FFH           ; test for timer active
        jr     nz,ERASETIMING         ; if the timer active jump
        clr     eraset                ; clear the erase timer

ERASETIMING:
        cp     eraset,#48D            ; test for the erase period
        jr     z,ERASETIME            ; if timed out the erase
        ret                            ; else we return

ERASETIME:
        or     P2,#00000010b          ; turn off the led
        ld     skipradio,#0FFH        ; set the flag to skip the radio read
        call   CLEARCODES            ; clear all codes in memory
        clr     skipradio             ; reset the flag to skip radio

        ld     learnt,#0FFH           ; set the learn timer
        ret                            ; return

ERASERELEASE:
        ld     eraset,#0FFH           ; turn off the erase timer
        ret                            ; return

INLEARN:
        cp     learndb,#20D           ; test for the debounce period
        jr     nz,TESTLEARNTIMER      ; if not then test the learn timer
        ld     learndb,#0FFH          ; set the learn db

TESTLEARNTIMER:
        cp     learnt,#240D           ; test for the learn 30 second timeout
        jr     nz,ERASETEST           ; if not then test erase

learnoff:
        or     P2,#00000010B          ; turn off the led
        ld     learnt,#0FFH           ; set the learn timer
        ld     learndb,#0FFH          ; set the learn debounce
        jr     ERASETEST              ; test the erase timer

```

```

.....
: WRITE WORD TO MEMORY
: ADDRESS IS SET IN REG ADDRESS
: DATA IS IN REG MTEMPH AND MTEMPL
: RETURN ADDRESS IS UNCHANGED
.....

```

# WRITEMEMORY:

```

push    RP                ; SAVE THE RP
srp     #LearnEeGroup     ; set the register pointer

call    STARTB            ; output the start bit
ld      serial,#00110000B ; set byte to enable write
call    SERIALOUT         ; output the byte
and     csport,#csl       ; reset the chip select
call    STARTB            ; output the start bit
ld      serial,#01000000B ; set the byte for write
or      serial,address    ; or in the address
call    SERIALOUT         ; output the byte
ld      serial,mtemp      ; set the first byte to write
call    SERIALOUT         ; output the byte
ld      serial,mtempl     ; set the second byte to write
call    SERIALOUT         ; output the byte
call    ENDWRITE          ; wait for the ready status
call    STARTB            ; output the start bit
ld      serial,#00000000B ; set byte to disable write
call    SERIALOUT         ; output the byte
and     csport,#csl       ; reset the chip select
pop     RP                ; reset the RP
ret

```

.....

```

; READ WORD FROM MEMORY
; ADDRESS IS SET IN REG ADDRESS
; DATA IS RETURNED IN REG MTEMPH AND MTEMPL
; ADDRESS IS UNCHANGED

```

.....

## ReadMemory:

```

push    RP                ;
srp     #LearnEeGroup     ; set the register pointer

call    STARTB            ; output the start bit
ld      serial,#10000000B ; preamble for read
or      serial,address    ; or in the address
call    SERIALOUT         ; output the byte
call    SERIALIN          ; read the first byte
ld      mtemp,serial      ; save the value in mtemp
call    SERIALIN          ; read teh second byte
ld      mtempl,serial     ; save the value in mtempl
and     csport,#csl       ; reset the chip select
pop     RP
ret

```

.....

```

; WRITE D CODE TO 4 MEMORY ADDRESS
; CODE IS IN Radio1P5H Radio1P5L RadioP5H RadioP5L
; CODE IS IN Count1P5H Count1P5L CountP5H CountP5L

```

.....

## WRITE\_D\_CODE:

```

push    RP                ;
srp     #LearnEeGroup     ; set the register pointer
ld      mtemp,RadioP5H    ; transfer the data

```

```

ld      mtempl, RadioP5L
call    WRITEMEMORY      ; write the temp bits
inc     address          ; next address
ld      mtempl, Radio1P5H
ld      mtempl, Radio1P5L
call    WRITEMEMORY      ; write the temps
inc     address          ; next address
STORE_COUNTER:
ld      mtempl, MirrorA
ld      mtempl, MirrorB
call    WRITEMEMORY      ; write the temps
inc     address          ; next address
ld      mtempl, MirrorC
ld      mtempl, MirrorD
call    WRITEMEMORY      ; write the temps
dec     address          ; reset the address
dec     address
dec     address
pop     RP
ret
; return

```

```

STORE_D_COUNTER:
push    RP
srp     #LearnEeGroup    ; set the register pointer
inc     address
inc     address
jr      STORE_COUNTER

```

```

;.....
; START BIT FOR SERIAL NONVOL
; ALSO SETS DATA DIRECTION AND AND CS
;.....

```

```

STARTB:
ld      P2M, #P2M_INIT    ; set port 2 mode
and     csport, #csl
and     clkport, #clockl
and     dioport, #dol
or      csport, #csh
or      dioport, #doh
or      clkport, #clockh
and     clkport, #clockl
and     dioport, #dol
ret
; set the chip select
; set the data out high
; set the clock
; reset the clock low
; set the data low
; return

```

```

;.....
; END OF CODE WRITE
;.....

```

```

ENDWRITE:
ld      P2M, #(P2M_INIT+1) ; set port 2 mode
and     csport, #csl
nop
or      csport, #csh
WDT
; reset the chip select
; delay
; set the chip select
; kick the dog
ENDWRITELOOP:
ld      temph, dioport
; read the port

```

```

and    temph,#doh          ; mask
jr     z,ENDWRITELOOP      ; if the bit is low then loop
and    csport,#csl         ; reset the chip select
ld     P2M,#P2M_INIT       ; set port 2 mode forcing output mode
ret

```

```

.....
; SERIAL OUT
; OUTPUT THE BYTE IN SERIAL
.....

```

```

SERIALOUT:

```

```

        ld     P2M,#P2M_INIT      ; set port 2 mode
        ld     templ,#8H          ; set the count for eight bits
SERIALOUTLOOP:
        rlc     serial            ; get the bit to output into the carry
        jr     nc,ZEROOUT         ; output a zero if no carry
ONEOUT:

```

```

        or     dioport,#doh       ; set the data out high
        or     clkport,#clockh    ; set the clock high
        and    clkport,#clockl    ; reset the clock low
        and    dioport,#dol       ; reset the data out low
        djnz   templ,SERIALOUTLOOP

```

```

        ret                     ; loop till done
                                ; return
ZEROOUT:

```

```

        and    dioport,#dol       ; reset the data out low
        or     clkport,#clockh    ; set the clock high
        and    clkport,#clockl    ; reset the clock low
        and    dioport,#dol       ; reset the data out low
        djnz   templ,SERIALOUTLOOP

```

```

        ret                     ; loop till done
                                ; return

```

```

.....
; SERIAL IN
; INPUTS A BYTE TO SERIAL
.....

```

```

SERIALIN:

```

```

        ld     P2M,#(P2M_INIT+1) ; set port 2 mode
        ld     templ,#8H          ; set the count for eight bits
SERIALINLOOP:

```

```

        or     clkport,#clockh    ; set the clock high
        rcf                                ; reset the carry flag
        ld     temph,dioport      ; read the port
        and    temph,#doh         ; mask out the bits
        jr     z,DONTSET
        scf                                ; set the carry flag
DONTSET:

```

```

        rlc     serial            ; get the bit into the byte
        and    clkport,#clockl    ; reset the clock low
        djnz   templ,SERIALINLOOP

```

```

        ret                     ; loop till done
                                ; return

```

.....  
**CLEAR PAGE 0 CODES IN THE MEMORY**  
 .....

**CLEARCODES:**

```

                                push    RP
                                di      ; disable interrupts
                                ld      SkipRadio,#0FFH
                                srp     #LearnEeGroup      ; set the register pointer
                                ld      Radio1P5H,#0FFH    ; set the codes to illegal codes
                                ld      Radio1P5L,#0FFH
                                ld      RadioP5H,#0FFH
                                ld      RadioP5L,#0FFH
                                clr     address
                                ld      ; set the page
                                cmp,#07d                    ; erase 7 values

ClearLoop:
                                call    WRITE_D_CODE
                                add     address,#4d        ; clear this address
                                djnz    cmp,ClearLoop      ; next clear address
                                clr     mtemph              ; clear data
                                clr     mtempl
                                ld      address,#1FH       ; set the address
                                call    WRITEMEMORY
                                pop     RP
                                ret      ; return
  
```

.....  
**TIMER UPDATE FROM INTERRUPT EVERY .256mS**  
 .....

**TimerOneInt:**

```

                                inc     TaskSwitch
                                ld      IMR,#RETURN_IMR    ; set to the next switch
                                tm      TaskSwitch,#00000001b ; turn on the interrupt
                                jr      nz,SkipRsRoutine    ; even odd
                                call    RS232               ; do rs232 .5 mS
                                ; do the serial

SkipRsRoutine:
                                tm      TaskSwitch,#00000011B ; test for task 0,1,2 or 3
                                jr      z,TASK1             ; task 1 every 1 mS

TASK0:
                                iret

TASK1:
                                push    RP

ONEMS:
                                srp     #LearnModeGroup    ; set the register pointer
                                inc     T4MS                ; increment the 4mS timer
                                inc     T125MS              ; increment the 125 mS timer
                                cp      T4MS,#4D           ; test for the time out
                                jp      nz,TEST125          ; if not true then jump

FOURMS:
                                clr     T4MS                ; reset the timer
                                cp      rto,#0FFh          ; test for the end of the rto
                                jr      z,RTOOK            ; if the radio timeout ok then skip
                                inc     rto                 ; increment the rto

RTOOK:
                                ei      ; enable the interrupts
  
```

	inc	mono	; increment the mono time out
	jr	nz, MONOOK	; if the mono timeout ok then skip
MONOOK:	dec	mono	; back turn
	cp	SwitchSkip, #00	; test for the skip switches command
	jr	nz, TEST125	
TESTSW1:			
	tm	P2, #00100000B	; test switch one
	jr	z, SW1SET	; if set jump
	cp	LearnDebounce, #00H	; test for min number
	jr	z, TESTSW2	; if at min skip dec
	dec	LearnDebounce	; dec debouncer down
SW1SET:	jr	TESTSW2	; next
	cp	LearnDebounce, #0FFH	; test for the max number
	jr	z, TESTSW2	; if at max skip inc
	inc	LearnDebounce	; inc the debouncer
TESTSW2:			
	tm	P2, #00000100B	; test switch two
	jr	z, SW2SET	; if set jump
	cp	CmdSwitch, #00H	; test for min number
	jr	z, TESTSWDB	; if at min skip dec
	dec	CmdSwitch	; dec debouncer down
SW2SET:	jr	TESTSWDB	; next
	cp	CmdSwitch, #0FFH	; test for the max number
	jr	z, TESTSWDB	; if at max skip inc
	inc	CmdSwitch	; inc the debouncer
TESTSWDB:			
TEST125:			
	cp	T125MS, #125D	; test for the time out
	jr	z, ONE25MS	; if true the jump
	pop	RP	
	iret		
ONE25MS:			
TOG:	ei		; enable the interrupts
	clr	T125MS	; reset the timer
	cp	SysDisable, #0FFH	; test for the top
	jr	z, DO12	
DO12:	inc	SysDisable	; count off the system disable timer
	cp	learnt, #0FFH	; test for overflow
	jr	z, LEARN1OK	; at roll over skip
LEARN1OK:	inc	learnt	; increase the learn timer
	cp	eraset, #0FFH	; test for overflow
	jr	z, ERASET1OK	; if at roll skip
ERASET1OK:	inc	eraset	; increase the erase timer
	pop	RP	

iret

## RS232 DATA ROUTINES

enter rs232 start with word to output in rs232do

RS232OSTART:

push	rp	; save the rp
srp	#TimerGroup	; set the group pointer
clr	RSStart	; one shot
ld	rs232odelay,#6d	; set the time delay to 3. mS
clr	rs232docount	; start with the counter at 0
and	RS232OP,#RS232OC	; clear the output
jr	NORSOUT	

RS232:

cp	RSStart,#0FFH	; test for the start flag
jr	z,RS232OSTART	

RS232OUTPUT:

push	rp	; save the rp
srp	#TimerGroup	; set the group pointer
cp	rs232docount,#11d	; test for last
jr	nz,RS232R	
or	RS232OP,#RS232OS	; set the output idle
JR	NORSOUT	

RS232R:

djnz	rs232odelay,NORSOUT	; cycle count time delay
inc	rs232docount	; set the count for the next cycle
scf		; set the carry flag for stop bits
rrc	rs232do	; get the data into the carry
jr	c,RS232SET	; if the bit is high then set
and	RS232OP,#RS232OC	; clear the output
jr	SETTIME	; find the delay time

RS232SET:

or	RS232OP,#RS232OS	; set the output
----	------------------	------------------

SETTIME:

ld	rs232odelay,#6d	; set the data output delay
tm	rs232docount,#00000001b	; test for odd words
jr	z,NORSOUT	; if even done
ld	rs232odelay,#7d	; set the delay to 7 for odd
		; this gives 6.5 * .512mS

NORSOUT:

RS232INPUT:

cp	rs232dicount,#0FFH	; test mode
jr	nz,RECEIVING	; if receiving then jump
tm	RS232IP,#RS232IM	; test the incoming data
jr	nz,NORSIN	; if the line is still idle then skip
clr	rs232dicount	; start at 0
ld	rs232idelay,#3	; set the delay to mid

RECEIVING:

djnz	rs232idelay,NORSIN	; skip till delay is up
------	--------------------	-------------------------

	inc	rs232dicount	; bit counter
	cp	rs232dicount,#10d	; test for last timeout
	jr	z,DIEVEN	
	tm	RS232IP,#RS232IM	; test the incoming data
	rcf		; clear the carry
	jr	z,SKIPSETTING	; if input bit not set skip setting carry
	scf		; set the carry
SKIPSETTING:			
	rrc	rs232di	; save the data into the memory
	ld	rs232delay,#6d	; set the delay
	tm	rs232dicount,#00000001b	; test for odd
	jr	z,NORSIN	; if even skip
	ld	rs232delay,#7	; set the delay
	jr	NORSIN	
DIEVEN:			
	ld	rs232dicount,#0FFH	; turn off the input till next start
	ld	rscommand,rs232di	; save the value
	clr	RSCount	; clear the counter
NORSIN:			
	pop	rp	; return the rp
	ret		
	Fill		
	Fill		
	Fill		
	Fill		
	Fill		
	Fill		
	Fill		
.end			